Django, tests et testabilité

Comment mettre en place une stratégie de tests

Difficultés pour tester

- Les tests utilisent le code d'une manière différente que le produit lui-même (interfaces, ...)
- Accès à des système externes pas toujours évident
- Lecture / Écriture de fichiers dangereuse
- Complexité du processus (comment je me branche là-dedans?)

Testabilité

- Un code testable suppose plus de tests, moins de bugs, un meilleur design...
- Toute action visant à améliorer la testabilité est bonne pour la qualité du code!
- Si l'on peut facilement extraire les librairies utilisées, il sera facile de les remplacer à l'avenir, et nos tests se focaliseront sur nos algorithmes.

Exemple

```
class PDFWriter:
    def drawstring(self, text): ...
    def endpage(self): ...

############ Non testable
class PDFReport(PDFWriter):
    def make_report(self):
        self.drawstring(self.title)
        for line in self.lines:
            self.drawstring(line)
        self.endpage()

############ Testable!!
class PDFReport:
    def __init__(self, writer=PDFWriter):
        self.pdf = writer()

    def make_report(self):
        self.pdf.drawstring(self.title)
        for line in self.lines:
            self.drawstring(line)
        self.endpage()
```

Mock?

- Un Mock simule de manière contrôlée le comportement d'une fonction ou d'un objet.
  - remplacement d'une librairie complexe
  - remplacement d'un système de fichiers
  - simulation de panne
  - données falsifiées
- Il permet de garder trace des appels effectués

Exemple

```
import smtplib
from email.mime.text import MIMEText

def send_email(body, engine=smtplib.SMTP)
    msg = MimeText(body)
    msg['subject'] = 'Important mail from admin'
    msg['from'] = 'admin@epfl.ch'
    msg['to'] = _get_all_users(some_param)
    sender = engine()
    sender.sendmail('admin@epfl.ch', 'All users', msg.as_string() )
    sender.quit()


from mock import Mock
def test():
    smtp_mock = Mock()
    send_email('test text'), engine=smtp_mock)
    assert(smtp_mock.mockCheckCall(0, 'sendmail', ...)
```

Les tests avec Django

- Django offre deux manières de tester:
  - Doctests: Les tests sont écrits dans la documentation des méthodes / modules. Seuls ceux disponibles dans les modèles seront utilisés.
  - unittest: les tests sont regroupés en scénarios (ex: un utilisateur soumet une publication à Infoscience), eux-mêmes séparés tests (ex: ajout d'un document), dans lesquels des assertions sont testées (ex: le fichier n'a pas changé de taille).

Tests unitaires

- Django crée une base de données vide pour chaque test.
  - Les données de production ne sont pas touchées
  - Aucun utilisateur n'est préchargé
  - Les données sont détruites après chaque test.
- Les tests sont extrêmement simples à lancer:
  - manage.py test [module[.scenario[.test]]]

Django - unittest

```
import os, tempfile
from django.test import TestCase
from auth import User

from monapp.utils import ma_fonction

class MonTest(unittest.TestCase):
    def setUp(self):
        self.user = User.objects.create(username='tester')
        self.directory = tempfile.mkdtemp()
        self.file = open(os.path.join(self.directory, 'testfile.txt')
        self.file.write('texte de test').close()

    def tearDown(self):
        os.remove(self.file)

    def test_montruc(self):
        self.assertTrue(ma_fonction(self.file) == 'texte_de_test')
```

Tester des vues

```
from django.test import Client
from django.core.urlresolvers import reverse
from monapp.models import Blog

client = Client() # self.client dans un test unitaire!
response = client.get(reverse('blog_index'))
assert(response.status_code == 200)

response = client.get(reverse('blog_article', {id: 1}))
assert(response.status_code == 404)
Blog.objects.create(id=1)
response = client.get(reverse('blog_article', {id: 1}))
assert(response.status_code == 200)

response = client.post(reverse('blog_post', {'title'='Test', body='test body'}))
assert(response.status_code == 403) # forbidden
assert(len(Blog.objects.filter(title='test')) == 0)

client.login(username='test', password='mot de passe') # l'utilisateur a été créé ailleurs
response = client.post(reverse('blog_post', {'title'='Test', body='test body'}))
assert(response.status_code == 200)
assert(len(Blog.objects.filter(title='test')) == 1)
```

Pour aller plus loin

- http://docs.djangoproject.com/en/dev/topics/testing/
- http://djangotesting.com/
- Tests and testability, par Ned Batchelder: (http://us.pycon.org/2010/conference/schedule/event/114/)
- http://python-mock.sourceforge.net/